Full abstraction for sequential languages : The states of the art

Gérard Berry, Pierre-Louis Curien, Jean-Jacques Levy

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

Rapports de Recherche

N° 197

FULL ABSTRACTION FOR SEQUENTIAL LANGUAGES : THE STATE OF THE ART

Gérard BERRY
Pierre Louis CURIEN
Jean-Jacques LEVY

Mars 1983

FULL ABSTRACTION FOR SEQUENTIAL LANGUAGES: THE STATE OF THE ART

G. Berry, Ecole des Mines, Sophia-Antipolis
P.L. Curien, LITP, Université Paris VII
J.J. Lévy, INRIA, Rocquencourt

Abstract

One usually considers two kinds of semantics for programming languages: the operational semantics, which describes an execution mechanism, and the denotational semantics, which associates input-output functions with program pieces. We survey the known results about the full abstraction problem for Plotkin's language PCF, i.e. the problem of finding denotational semantics which agree completely with the operational ones. After giving basic syntactic results on PCF, we recall Milner's results on the existence and uniqueness of a fully abstract model. Milner's construction of that model is purely syntactic, and the problem of its semantic characterization is still open. We survey the known semantic models, obtained from order-enriched cartesian closed categories: continuous and stable functions, sequential algorithms.

Résumé

Il existe deux types principaux de sémantiques des langages de programmation: les sémantiques opérationnelles, qui décrivent des mécanismes d'évaluation, et les sémantiques dénotationnelles, dans lesquelles on associe des fonctions d'entrée-sortie aux parties d'un programme. Nous des sémantiques du langage PCF de Plotkin, i.e. le problème de la correspondance entre les sémantiques opérationnelles et dénotationnelles. Nous rappelons d'abord les principaux résultats syntaxiques sur PCF, issus de la théorie du lambda-calcul. Nous donnons ensuite les résultats de Milner sur l'existence et l'unicité du modèle complètement adéquat, qui est obtenu de manière purement syntaxique. Sa caractérisation sémantique est un problème ouvert. Nous étudions les modèles sémantiques connus, obtenus à partir de catégories cartésien fermées: fonctions continues, fonctions stables, algorithmes séquentiels.

This paper will appear in the Acts of the French-American Seminar on Semantics and should be cited as such.

Ce papier paraîtra dans les Actes du Colloque Franco-Américain de Sémantique, Fontainebleau, 1982, et doit être cité comme tel.

1. Introduction.

When writing programs we usually think in terms of rather abstract entities: numbers, trees, functions... But when running programs we use machines for which these entities make no sense: they only push symbols. We generally hope that there is a reasonable correspondence between what we want to "compute" and the list of symbols output by the machine at the end of the execution (if any). This resembles the classical relation between axiomatic theories and models in logic: operational semantics (what the machine does) corresponds to axiomatic deduction, while abstract semantics corresponds to models. Hence the required correspondence should be expressed by *adequation* and *completeness* results. In computer science, contrarily to what happens in classical logic, the primary concept is not that of model, but that of machine (the machine is always right). Hence adequation and completeness will be defined according to a machine, or in more accurate terms to an operational semantics.

We shall focus our attention on the problem of *operational equivalence and ordering* of program pieces. Following a quite common terminology, we shall call *program* any object which is directly executable and produces observable results, and *procedure* any (sensible) piece of text which may serve as a part of a program, without being executable on its own. As in [32, 34] we say that two procedures M_1 and M_2 are *operationally equivalent* if they are interchangeable in any program context, i.e. if the result of any program does not change when replacing one by the other, and more generally that M_1 *is operationally less defined* than M_2 if any programs which terminates using M_1 terminates with the same result using M_2. The operational equivalence and ordering are sensibly defined as soon as the notion of result of a program is, and are clearly very natural and essential as far as execution of programs is concerned. But in order to reason about programs, one always prefer to consider procedures as denoting *functions from their inputs to their outputs*, and to use any kind of mathematical techniques to compare such functions. The classical *denotational semantics* techniques [40] provide us with an adequate mathematical framework for studying ordering and equality of such functions. Given M_1 and M_2 as above, we say that M_1 is *denotationally less defined* than M_2 if the function defined by M_1 is less than the function defined by M_2 in the appropriate function space. The correspondence problem between operational and denotational semantics may then be precisely stated: we say that a semantics is *adequate* if the

denotational ordering implies the operational one, *complete* if the converse holds. Following Plotkin [34] and Milner [32] we say that a semantics is *fully abstract* when it is adequate and complete. In adequate semantics, denotational proofs of equivalence are operationally valid. But proofs of difference are valid only in fully abstract semantics.

Of course the stated problem is very general and applies to all kinds of programming languages. We shall study it only for *sequential languages*, without trying to define precisely that term (for fixing ideas, PASCAL, LISP are sequential, while languages which admit synchronization primitives are not). See [1, 20] for other kinds of languages, including parallel ones. In the sequential case, the problem was originally raised and shown difficult by Plotkin [34], and further studied by Milner [32] and by the authors [6, 7, 11, 12, 19]. It is still unsolved. However many new results have been obtained on the way, both on syntax and semantics, with side effects useful to other problems. Our purpose here is to give a survey of the results, with indications about their proofs. Our presentation of some results may differ from their original one, since we use as much as possible strong syntactic properties which were not known at the time the results were originally found.

We shall use a language introduced by Plotkin [34] and called *PCF* (Programming Computable Functions). It is a typed lambda-calculus with arithmetic primitives, and to our opinion has two advantages: it is syntactically simple, it is general enough to raise almost all the possible problems in the field. The *operational semantics* of *PCF* is not defined in term of a "machine", but by a set of *rewrite rules*. Some of the rules concern parameter passing for procedures: they are the usual (α) and (β) rules of the lambda calculus. One rule concerns recursion done via the fixpoint operator Y. The remaining rules concern arithmetic and boolean computations, and are purely first order. The use of rewrite rules for operational semantics has many advantages, in particular the great advantage of relying on fairly well-known syntactic theories such as the lambda calculus [3] or the theory of first order rewriting systems [24]. (Recently Plotkin [35] showed that rewrite rules are an excellent tool for describing the operational semantics of arbitrarily complex languages.) The syntax and operational semantics of *PCF* are given in section 2.

In section 3, we analyze the syntactic properties of *PCF*. Using inductions based on a finite termination property, we show the Church-Rosser and standardization theorems. Then we introduce Boehm trees and show the

syntactic continuity theorem. We finish with the sequentiality and stability theorems, which express the sequentiality property of *PCF*. This set of results is fairly classical in the lambda calculus area, see for example [3, 14]. However we try as much as possible to combine similar but independent results on the lambda calculus and on first order term rewriting systems, in order to obtain global results for *PCF*. We indicate how to generalize this process to similar languages, using in particular the results of [23].

Section 4 discusses *models*. We first define *interpretations* of first-order symbols, then least fixpoint models of interpretations. We show that any least fixpoint model is adequate.

In section 5 we construct models from syntax. We recall Milner's essential result [32]: there exists a unique (extensional) fully abstract model of *PCF*, which is characterized by the definability of all its (isolated) points. The model is constructed from a suitable completion of term operational equivalence classes. We also construct a fully abstract model of the initial (or free, or Herbrand) interpretation, where function symbols act just as term constructors. We show that the previous uniqueness property of the fully abstract model does not hold in that case: in fact any model of the free interpretation is fully abstract.

Constructing fully abstract models from syntax is clearly not completely satisfactory, since it gives no information on how to reason about programs. The problem is now to construct models in a *semantic* way, i.e. by using concepts not directly related to the language itself. The natural framework for semantic models is that of *cpo-enriched cartesian closed categories* [7, 10, 29, 37]. We show in section 6 that any such category yields a least fixpoint model.

The simplest such category is of course that of cpo's and continuous functions. Plotkin [34] showed that the obtained model is not fully abstract, since it contains non definable objects such as the "parallel or" *por*, which yields true as soon as one of its argument is true. We use the sequentiality theorem to show that *por* is not definable. We also recall Plotkin's result that adding *por* (in fact a similar "parallel conditional") to the syntax of *PCF* makes the continuous function model fully abstract. However the language obtained is not anymore sequential, and this result gives no information about our original problem. We also study full subcategories of continuous functions, and in particular the "concrete" category of event structures introduced by Winskel [46].

The continous function model being too big, we must find more restricted classes of functions, and if possible a good notion of "sequential" function. A

first interesting approximation is that of *stable function* introduced in [4, 6, 7]. The stability constraint excludes functions such as *por*, and stable functions form a suitable category. An important fact is that stable function may *not* be ordered by the classical pointwise ordering of continuous functions. We have to introduce another ordering for them, the *stable ordering* $\leq_s$. Since we know from Milner's result that the fully abstract model is ordered pointwise, the stable model is not fully abstract. In a second step, we construct a finer category by keeping together *both* orderings in he construction. The model is then much closer to the fully abstract model, but still contains non definable elements. We show that the stable ordering is also present in syntax, and that the fully abstract model is indeed a biordered model. We conjecture that the stable ordering is the exact image of the Boehm tree ordering of terms.

Sequentiality is more complex, and is studied in detail in another paper in this volume [11]. We recall the main results: a fairly general definition of sequentiality has been given by Kahn and Plotkin in their study of concrete data structures [25]. But Kahn-Plotkin sequential functions do *not* form a cartesian closed category [12]. However one can sensibly forget about functions, and construct a model from a new notion of *sequential algorithm* [12, 11, 17]. That model gives deep insights into the nature of sequential computation, and is fully abstract for another sequential language, the language *CDS* of [9, 11, 19]. But it is not fully abstract for *PCF* (although it is indeed fully abstract with respect to a richer notion of procedure observation that we do not discuss here). We are currently trying to do with algorithms what we did with stable functions: keep the pointwise ordering in the model construction. Winskel's results on *stable event structures* [46] should be of great interest there, see for example [18]. There are still serious difficulties, and it is not certain that such constructions will yield the fully abstract model.

2. The language *PCF* and its operational semantics.

2.1. Syntax of *PCF*.

The language *PCF* is a typed lambda calculus with arithmetic operators. We assume basic knowledge of lambda calculi, as described in [3].

We consider two *ground types* ι (integers) and o (booleans). The set T of *types* is the least set containing ι, o, and $(\sigma \to \tau)$ for $\sigma, \tau \in T$. We write

$$\sigma_1 \times \sigma_2 \times \cdots \times \sigma_n \to \tau = (\sigma_1 \to (\sigma_2 \to (\cdots \to (\sigma_n \to \tau)\ldots)))$$

We define the following *PCF* typed *constants*:

$\underline{n}$: ι for any integer n

$\underline{tt}, \underline{ff}$: o (truth values)

$\underline{+1}, \underline{-1}$: $\iota \to \iota$ (successor and predecessor)

$\underline{0?}$: $\iota \to o$ (test to zero)

$\underline{\text{if}}_\iota$: $o \times \iota \times \iota \to \iota$ (integer conditional)

$\underline{\text{if}}_o$: $o \times o \times o \to o$ (boolean conditional)

Let V^σ be a denumerable set of *variables* of type σ for each σ, let $V = \cup_\sigma V^\sigma$. The set $PCF = \cup^\sigma PCF^\sigma$ is defined by:

(i) $\Omega^\sigma \in PCF^\sigma$ for any σ

(ii) $x^\sigma \in PCF^\sigma$ for any variable x^σ

(iii) $f^\sigma \in PCF^\sigma$ for any PCF constant f of type σ

(iv) If $M^{(\sigma \to \tau)} \in PCF^{(\sigma \to \tau)}$ and $N^\sigma \in PCF^\sigma$, then $(M^{(\sigma \to \tau)} N^\sigma) \in PCF^\tau$

(v) If $x^\sigma \in V^\sigma$ and $M^\tau \in PCF^\tau$, then $(\lambda x^\sigma M^\tau) \in PCF^{(\sigma \to \tau)}$

(vi) If $M^{(\sigma \to \sigma)} \in PCF^{(\sigma \to \sigma)}$, then $YM^{(\sigma \to \sigma)} \in PCF^\sigma$

Our version of *PCF* differs from that of Plotkin [34] since the "syntactic undefined" Ω belongs to the language, and since the fixpoint operator Y cannot be used as a constant. This simplifies many results without loss of generality (one can define a fixpoint constant YY by $YY = \lambda x.Yx$). It also differs from Milner's version [32] which is based on combinatory logic.

As usual we shall omit types whenever possible, and use left association for application and right association for λ's, so that $\lambda xy.xyz$ stands for $(\lambda x(\lambda y((xy)z)))$. The notions of *free* and *bound* occurrences of variables are standard, and we call $M[N/x]$ the result of the substitution of all free occurrences of x by N in M (x and N must have the same type). We shall never care about α-conversion, and identify $\lambda x.M$ and $\lambda y.M[y/x]$ (y not free in M). A *closed term* is a term without free variables. We shall also use *contexts*, which are expressions C^σ containing occurrences of *holes* $[]^\tau$. By filling a hole with a term M of appropriate type, one obtains a new term $C[M]$. Notice that this operation may bind free variables of M by binders of $C[]$, as in $\lambda x.[x]$; such a capture is not possible by substitution alone. We also consider contexts with several holes $[]_i$; given any vector $\overline{M}$ of terms we call $C[\overline{M}]$ the term obtained by filling $[]_i$ with M_i.

We define the Ω-*match ordering* $\leq_\Omega$ on terms as the least ordering such that $\Omega^\sigma \leq_\Omega M^\sigma$ for all M^σ and such that $M \leq_\Omega N$ implies $C[M] \leq_\Omega C[N]$ for all $M, N, C[]$. Clearly $M \leq_\Omega N$ holds iff N may be obtained by replacing some occurrences of Ω in M by *PCF* terms.

We shall also need to manipulate precisely *occurrences* in terms. The best way to understand the notion of occurrence is to write terms as trees and to define occurrences as paths in trees. An occurrence w is a word on $\{1,2\}$ and it determines a *subterm* $M\downarrow w$ of a term M at w. The set $OCC(M)$ of occurrences in M is defined as follows:

(i) For any M, the empty word ε belongs to $OCC(M)$ with $M\downarrow\varepsilon = M$.

(i) If M is a variable, a constant or an Ω, then ε is the only occurrence in M

(ii) $OCC(M_1 M_2) = \{\varepsilon\} \cup \{w1 \mid w \in OCC(M_1)\} \cup \{w2 \mid w \in OCC(M_2)\}$
with $M\downarrow w1 = M_1\downarrow w$ and $M\downarrow w2 = M_2\downarrow w$.

(ii) $OCC(\lambda x.M) = OCC(YM) = \{\varepsilon\} \cup \{w1 \mid w \in OCC(M)\}$
with $\lambda x.M\downarrow w1 = YM\downarrow w1 = M\downarrow w$.

2.2. Operational semantics of PCF.

Here are the *reduction rules* of PCF:

(succ) $\underline{+1}\ \underline{n} \to \underline{n+1}$
(pred) $\underline{-1}\ \underline{n+1} \to \underline{n}$
(zero1) $\underline{0?}\ \underline{0} \to \mathit{tt}$
(zero2) $\underline{0?}\ \underline{n+1} \to \mathit{ff}$
(cond1) $\underline{\mathrm{if}}\ \mathit{tt}\ x\ y \to x$
(cond2) $\underline{\mathrm{if}}\ \mathit{ff}\ x\ y \to y$
(cond3) $\underline{\mathrm{if}}_\bullet\ \mathit{tt}\ x\ y \to x$
(cond4) $\underline{\mathrm{if}}_\bullet\ \mathit{ff}\ x\ y \to y$
(beta) $(\lambda x.M)N \to M[N/x]$
(Y) $YM \to M(YM)$
(cont) $M \to N$ implies $C[M] \to C[N]$ for any context $C[]$

Notice that the relation $\to$ preserves types. We call $\overset{*}{\to}$ its reflexive and transitive closure. Given any reduction $M \to N$ there exists a unique context $C[]$ and unique terms M_1, N_1 such that $M = C[M_1]$, $N = C[N_1]$, and $M_1 \to N_1$, this reduction not being of type (cont). Then M_1 is called the *redex* of the reduction. If w is the corresponding occurrence of M_1 in M, we write $M \overset{w}{\to} N$. To be perfectly correct, one should always name the redex occurrences in reductions and

distinguish the notion of reduction - sequence of labelled reduction steps - from the reduction relation.

A *program* P is a closed term of type o. We say that P *computes* n if $P \stackrel{\bullet}{\to} \underline{n}$ (we shall see in a moment that P computes at most one n). If $P \stackrel{\bullet}{\to} \underline{n}$ holds for no n, then P is said to *diverge;* the computation of P may then be infinite as for $P = Y(\lambda x.x)$, or it may reach terms which have no further reductions but are not integer constants, as for $P = \underline{-1}\,\underline{0}$.

Let M, N be two terms of the same type. One says that M is *operationally less defined than* N and one writes $M \leq_{op} N$ if $P[M] \stackrel{\bullet}{\to} \underline{n}$ implies $P[N] \stackrel{\bullet}{\to} \underline{n}$ for all context $P[]$ such that $P[M]$ and $P[N]$ are both programs (such a context will often be called a *program context* for M and N). We say that M and N are *operationally equivalent* and write $M =_{op} N$ if $M \leq_{op} N$ and $N \leq_{op} M$.

The relations $\leq_{op}$ and $=_{op}$ correspond to the intuition given in the introduction: $M \leq_{op} N$ holds iff N computes more than M in any program context, $M =_{op} N$ holds iff M and N may be freely interchanged in any program context. To understand their properties, we must investigate the structure of reductions. Notice that it is not even obvious that $P \stackrel{\bullet}{\to} \underline{n}$ and $P' \stackrel{\bullet}{\to} \underline{n}$ imply $P =_{op} P'$!

3. Syntactic properties of PCF.

We investigate the main syntactic properties needed in the sequel: Church-Rosser, standardization, continuity, stability, sequentiality.

3.1. λ- and δ-rules.

It is essential to notice that the reduction rules of PCF are of two quite different kinds: the rules (beta), (Y) and (cont), which we call λ-rules $\stackrel{}{\to}_{\lambda}$, are not concerned with the particular nature of the *PCF* constants, and serve for manipulating higher-order expressions (i.e. procedures); on the contrary the other rules serve only for computing ground values, and have actually nothing to do with the lambda-calculus, being purely first-order. We call them δ-rules $\stackrel{}{\to}_{\delta}$. There are many reasons to study λ- and δ-rules separatedly: the λ-rules are very general and are useful with many other sets of δ-rules, and we shall even pay much attention to the case where there are no δ-rules at all; the techniques involved for studying both kinds of rules are slightly different (actually one may hope to develop a very general set of results overcoming this difficulty, as

suggested by the results of [23], but this is far beyond the scope of this paper); last but not least, the following simple commutation result shows that the application of δ-rule can be delayed arbitrarily:

3.1.1. **Lemma:** If $M_1 \underset{\delta}{\rightarrow} M_2 \underset{\lambda}{\rightarrow} M_3$, there exists N such that $M \underset{\lambda}{\rightarrow} N \overset{*}{\underset{\delta}{\rightarrow}} M_3$.

proof: easy case inspection.

3.1.2. **Proposition:** If $M_1 \overset{*}{\rightarrow} M_2$, there exists N such that $M_1 \overset{*}{\underset{\lambda}{\rightarrow}} N \overset{*}{\underset{\delta}{\rightarrow}} M_2$.

proof: Easy induction using the previous lemma.

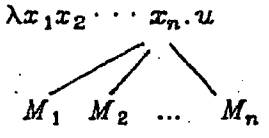
As far as operational semantics is concerned, we can even go a step further. We are mainly interested in reductions of the form $P \overset{*}{\rightarrow} n$, which may therefore be written $P \overset{*}{\underset{\lambda}{\rightarrow}} Q \overset{*}{\underset{\delta}{\rightarrow}} n$. It is quite clear that the λ's and Y's of Q are of no use in the reduction $Q \overset{*}{\underset{\delta}{\rightarrow}} n$, and that all the subterms of Q not concerned with the δ-reduction may be simply replaced by Ω's (consider for example $\underline{\text{if}}\ tt\ n\ YM$, which behaves exactly as $\underline{\text{if}}\ tt\ n\ \Omega$). The interest is that the term Q' so obtained belongs to the first-order algebra of *PCF* constants, so that the separation between first-order and higher-orders is made complete. The process of replacing subterms by Ω's being a very general and useful one, we shall define it for arbitrary terms. It is convenient to use Wadsworth's notion of head normal form [43]:

3.1.3. **Definition:** A term is in *head normal form* or *hnf* if it has the form $\lambda x_1 x_2 \cdots x_m . u M_1 M_2 \cdots M_n$, $m,n \geq 0$, u a variable or a constant. The *immediate syntactic value* $\omega(M)$ of M is defined as follows:

(i) $\omega(M)=\Omega$ if M is not in hnf;

(ii) $\omega(M)=\lambda x_1 x_2 \cdots x_m . u\ \omega(M_1)\omega(M_2) \cdots \omega(M_n)$
if $M=\lambda x_1 x_2 \cdots x_m . u M_1 M_2 \cdots M_n$.

It is often useful to use a tree representation for head normal forms and syntactic values:



The set of syntactic values is included in *PCF*, and is therefore also ordered by $\leq_\Omega$. We call it *BT* (for Boehm Trees, see [3, 7, 42]) and call its elements $t, t', \ldots$.

Coming back to programs, their syntactic values are simply the closed terms in the free ordered algebra generated by the set F of PCF constants: they contain only Ω's or constants. They form a set $A(F)$ or simply A, with elements called $a, a', \ldots$.

3.1.4. Proposition: Let P be a program. if $P \overset{*}{\to} n$, there exists Q such that $P \overset{*}{\underset{\lambda}{\to}} Q$ and $\omega(Q) \overset{*}{\underset{\delta}{\to}} n$

proof: Easy corollary of 3.1.2. using an induction on the size of $\omega(Q)$.

Hence we shall use λ-rules in PCF and δ-rules in $A(F)$. We call the corresponding calculi PCF^{λ} and A^{δ}. More generally we shall be able to talk about languages combining the lambda-calculus and a set of first-order rules.

3.2. Finite Termination Properties.

In the study of term rewriting systems, it is well known that the Finite Termination Property or FTP is extremely useful, since it makes it possible to use a single induction mechanism for proving all sorts of syntactic properties (a rewriting system is said to satisfy the FTP if it does not contain infinite reductions). It is also well known that the typed lambda calculus satisfies the FTP [3]. However PCF does not satisfy it, since terms such as $Y(\lambda x.x)$ have infinite reductions. As in [27, 28], we introduce a labelled PCF calculus PCF_l. It is constructed in the same way as PCF, except that Y combinations receive an integer label, which acts as a limit to the number of possible Y reductions. The formation rules of PCF_l are those of PCF, except the rule for Y which is now

(vi_l) If M is a PCF_l term of type $(\sigma \to \sigma)$ and if m is an integer, then $Y^m M$ is a PCF_l term.

The (Y) rule becomes

(Y_l) $Y^{m+1} M \to Y^m M$

3.2.1. Theorem: PCF_l^{λ} has the Finite Termination Property.

proof: Extension of the classical proof for the typed λ-calculus, see [7].

We shall refer to this property as the FTPL property of PCF^{λ}.

Moreover the calculus PCF_l^{λ} "simulates" PCF^{λ}. Given any PCF^{λ} reduction starting from a term M, it is always possible to label the occurrences of Y in M by integers in such a way that the original reduction can also be performed in PCF_l^{λ} (choose for example the length of the original reduction as the value of all labels). Now given a property we want to prove for PCF reductions of a given

term, say for example the Church Rosser property, we choose labels big enough for all reductions which interest our problem to be simulated in PCF_l^λ, and we prove the property in PCF_l^λ by induction on the length of the longest labeled reduction. See [7] for details. This induction technique will be called FTPL induction in the sequel.

The calculus A^δ evidently satisfies the FTP: the size of a term is decreased by a reduction. Generally speaking there exist good proof techniques for showing the FTP for first-order systems, see [24]. It is also not hard to see that the full PCF_l calculus satisfies FTP. We say simply that PCF satisfies FTPL.

3.3. The Church Rosser theorem.

The well-known *Church Rosser* or *confluence* property expresses as follows for PCF^λ:

3.3.1. Theorem: If $M \overset{*}{\underset{\lambda}{\to}} M_1$ and $M \overset{*}{\underset{\lambda}{\to}} M_2$, then there exists N such that $M_1 \overset{*}{\underset{\lambda}{\to}} N$ and $M_2 \overset{*}{\underset{\lambda}{\to}} N$.

It follows by a simple FTPL induction from the following "permutation lemma":

3.3.2. Lemma: If $M \underset{\lambda}{\to} M_1$ and $M \underset{\lambda}{\to} M_2$ then there exists N such that $M_1 \overset{*}{\to} N$ and $M_2 \overset{*}{\to} N$.

proof: very tedious case inspection.

The Churh Rosser property also holds for the full PCF calculus: it is easy to show it for the δ-rules as in [24] and to combine the results for λ- and δ-rules.

Remark: One may also give much stronger forms of the Church-Rosser property by considering the *permutation equivalence* of reductions [13, 14, 28], also called *strong equivalence* in [3], which is defined exactly as for the pure lambda-calculus. One then shows that reductions form an upper-semilattice, the upperbound of two reductions being obtained as in the usual Church-Rosser construction. Again this may be done both in PCF^λ and in A^δ, and extended to the full PCF.

Of course the Church-Rosser property establishes the soundness of our operational semantics:

3.3.3. Proposition: For any program P there exists at most one n such that $P \overset{*}{\to} n$.

3.4. The standardization theorem.

The standardization theorem is probably the most important technical tool for proving syntactic properties of languages such as *PCF*. It expresses that the rewritings may always be performed from left to right. Formally, let w, w' be two occurrences in M. One says that w is *on the left* of w' if either w is prefix of w' or there exists u, w_1, w'_1 such that $w = u1w_1$ and $w' = u2w'_1$. A reduction $M \overset{w_1}{\to} M_1 \overset{w_2}{\to} \cdots \overset{w_n}{\to} M_n$ is *standard* if w_i is on the left of w_{i+1} for any i such that $1 \leq i < n$.

3.4.1. Theorem: If $M \overset{*}{\to} M'$, then there exists a standard reduction from M to M'.

proof: by FTPL induction and induction on the size of M.

One can actually say more: given any reduction from M to M', there exists a unique strongly equivalent standard reduction [3, 14]. Hence standard reduction may serve as canonical elements of strong equivalence classes.

An analoguous *result* exists in A^{δ}, although the notion of "left to right reduction" works there by accident, for δ rules such as the rules for if also have a left to right character. This need not be true in general. See [23] for a general definition of standard reductions in first-order term rewriting systems.

The result extends to the full *PCF* calculus, but we shall use it only for PCF^{λ}.

3.5. Infinite Boehm trees and the continuity theorem.

The notion of Boehm tree introduced in 3.1. allows us to extend the notion of normal form (term having no further reduction) to infinite objects. The (classical) idea is to notice that the immediate syntactic values of terms increase along reductions, and to associate with any term the upperbound of the immediate syntactic values of its derivatives, which is nothing but an infinite Boehm tree. Infinite Boehm trees are a fundamental tool in the study of the operational semantics and also of the denotational one.

3.5.1. Definition: The set BT^{∞} of infinite Boehm trees is the order completion of the set BT ordered by $\leq_{\Omega}$ (see 3.1.). Intuitively, elements of BT may be seen as infinite trees of the form given in 3.1. The set A^{∞} is the order completion of A. It is then the set of infinite trees written with constants and ground Ω's.

3.5.2. Lemma: If $M \underset{\lambda}{\to} N$ then $\omega(M) \leq_{\Omega} \omega(N)$.

3.5.3. **Definition:** we set $BT(M)=lub\{\omega(N)\,|\,M\to^{*}N\}$. We set $M\leq_{BT}N$ if $BT(M)\leq_{\Omega}BT(N)$.

3.5.4. **Lemma:** If $M\underset{\lambda}{\to}N$ then $BT(M)=BT(N)$.

proof: By 3.5.2 and the Church-Rosser property.

The *syntactic continuity theorem* expresses that the context operation is continous w.r.t. Boehm trees:

3.5.5. **Theorem:** For any context $C[]$ and term vector $\overline{M}$, one has $BT(C[\overline{M}]) = lub\{BT(C[\overline{t}]) \mid \overline{t}\leq_{\Omega}BT(\overline{M})\}$.

proof: Several different proofs exist, see [3, 7, 27, 43, 45]. Once FTPL induction is known, the simplest proof is that of [27, 28].

As a corollary, one sees that the rules of formation of terms are monotonic w.r.t. Boehm trees:

3.5.6. **Corollary:** If $\overline{M}\leq_{BT}\overline{N}$ then for any context $C[]$ one has $C[\overline{M}]\leq_{BT}C[\overline{N}]$.

Remembering that PCF itself is ordered by the Ω-match ordering, we see that the function BT iself is monotonic from PCF to BT^{∞}:

3.5.7. **Corollary:** If $M\leq_{\Omega}N$, then $M\leq_{BT}N$

proof: Consider any term as the context of its Ω's.

Because of the continuity theorem, Boehm trees may be considered as first-class citizens. It is perfectly possible to put them into contexts, hence to apply and abstract them:

3.5.8. **Definition:** Given a context $C[]$ and a Boehm tree vector $\overline{T}$ in BT^{∞}, we set $C[T] = lub\{C[t] \mid t\ finite,\ t\leq_{\Omega}T\}$. In particular we set $T_1T_2=[T_1]_1[T_2]_2$ and $\lambda x.T=\lambda x.[T]$.

Of course the Boehm trees of programs contain only constants and ground Ω's, i.e. belong to A^{∞}. The following monotonicity property of $\overset{*}{\to}_{\delta}$ is obvious:

3.5.9. **Lemma:** Let a, a' in A such that $a\leq_{\Omega}a'$. If $a\overset{*}{\to}_{\delta}n$ then $a'\overset{*}{\to}_{\delta}n$.

From all these results we deduce an important property of program contexts:

3.5.10. **Theorem:** If $P[M]\overset{*}{\to}n$ and $M\leq_{\Omega}N$ then $P[N]\overset{*}{\to}n$.

proof: by 3.5.5, 3.5.7 and 3.5.8.

3.6. Stability and sequentiality of PCF.

Intuitively *PCF* has a "sequential" behaviour, its interpreter may be a "sequential" program (of course this does not mean that no parallel evaluation of PCF is possible; it just means that a natural sequential one exists). However the notion of sequentiality is not so easy to define correctly, and we shall actually not do it. We shall simply exhibit some syntactic properties which express in some way sequentiality. These properties will be essential to show later on that "non sequential" models are not fully abstract. As usual we shall combine two analoguous properties, one for λ-rules and one for δ-rules. The properties of λ-rules will be expressed in terms of the context operation, while the properties of δ-rules will concern reductions of first-order terms to integer constants. They will imply together properties of reductions $P[M]\overset{*}{\to}n$ using 3.1.4, i.e. properties useful for studying the relation $\leq_{op}$. The properties will not be given in their full generality, i.e. with infinite Boehm trees placed in contexts as in [5, 7].

The sequentiality theorem expresses that the function BT is sequential in the sense of Kahn-Plotkin [12, 25]. It establishes a relation between the occurrences of Ω in $BT(M)$ and in M.

3.6.1. Notation: We define occurrences for finite and infinite Boehm trees exactly as for ordinary terms.

3.6.2. Theorem: Let M be a term, let $T=BT(M)$, let u be an occurrence of Ω in T. Then either $BT(N)\downarrow u=\Omega$ for all N such that $M\leq_{\Omega}N$, or there exists an occurrence w of Ω in M such that $M\leq_{\Omega}N$ and $BT(N)\downarrow u\neq\Omega$ imply $N\downarrow w\neq\Omega$.

proof: Consequence of the standardisation theorem, see [3, 5, 7].

Thus in order to increase $BT(M)$ at occurrence u, it is *necessary* to increase M at occurrence w. The analoguous first-order property expresses as follows:

3.6.3. Proposition: Let t in A be such that $t\overset{*}{\to}n$ holds for no n. Then either $t'\overset{*}{\to}n$ holds for no t' such that $t\leq_{\Omega}t'$ and no n, or there exists an occurrence w of Ω in t such that $t\leq_{\Omega}t'$ and $t'\overset{*}{\to}n$ imply $t'\downarrow u\neq\Omega$

proof: by induction on the size of t.

Combining both results and using 3.1.4, we obtain:

3.6.4. Theorem: Let P be a program such that $P \stackrel{\bullet}{\to} n$ holds for no n. Then either $P' \stackrel{\bullet}{\to} n$ holds for no P' such that $P \leq_\Omega P'$ and no n, or there exists an occurrence w of Ω in P such that $P' \stackrel{\bullet}{\to} n$ implies $P'/w \neq \Omega$ for all P' such that $P \leq_\Omega P'$.

Hence it is *necessary* to increase P at occurrence w in order to produce some integer result. The occurrence w of Ω will be called a *sequentiality index*. The same results holds of course for boolean programs, and as a typical application, let us show that no "parallel or" combinator (i.e. closed term) can be defined in PCF. Such a combinator M would satisfy

$$M\ \mathit{tt}\ \Omega \stackrel{\bullet}{\to} \mathit{tt}$$

$$M\ \Omega\ \mathit{tt} \stackrel{\bullet}{\to} \mathit{tt}$$

$$M\ \mathit{ff}\ \mathit{ff} \stackrel{\bullet}{\to} \mathit{ff}$$

Then by the continuity theorem $M\ \Omega\ \Omega$ cannot reduce to tt or ff, since one has $M\ \mathit{tt}\ \Omega \stackrel{\bullet}{\to} \mathit{tt}$ and $M\ \mathit{ff}\ \mathit{ff} \stackrel{\bullet}{\to} \mathit{ff}$. But the term $M\ \Omega\ \Omega$ has then no index, which is impossible.

The second property is called the *stability property*. It is a consequence of the sequentiality property, but expresses quite differently and will be very useful for models. It says that the function BT is stable, see section 8. Think again of the continuity theorem: given a term M and an approximation T of $BT(M)$, what can we say about the set of approximations M' of M such that $T \leq_\Omega BT(M')$? The stability theorem asserts that it has a least element.

3.6.5. Theorem: Let M be a term, let $T \leq_\Omega BT(M)$. There exists a least $M' \leq_\Omega M$ such that $T \leq_\Omega BT(M')$.

proof: Easy from the sequentiality theorem 3.6.2.

The analoguous property of δ-rules expresses as follows:

3.6.6. Proposition: Let t be a finite term of A such that $t \stackrel{\bullet}{\to} n$. Then there exists a least $t' \leq_\Omega t$ such that $t' \stackrel{\bullet}{\to} n$.

proof: easy from 3.6.3.

By combining the results we obtain:

3.6.7. Theorem: Assume $P \overset{\bullet}{\to} \pi$. Then there exists a least $P' \leq_\Omega P$ such that $P' \overset{\bullet}{\to} \pi$.

We could also use the stability theorem for showing that no "parallel or" is definable: with the same notation as above, we see that there is no least approximation P' of $M\ tt\ tt$ such that $P' \overset{\bullet}{\to} tt$. But the stability theorem is weaker than the sequentiality theorem. We cannot use it for showing that no combinator M having the following properties can exist:

$$M\ tt\ ff\ \Omega \overset{\bullet}{\to} tt$$

$$M\ \Omega\ tt\ ff \overset{\bullet}{\to} tt$$

$$M\ ff\ \Omega\ tt \overset{\bullet}{\to} tt$$

One needs to apply the sequentiality theorem and to see that the term $M\ \Omega\ \Omega\ \Omega$ has no index. This example will be reused later on when studying stable models.

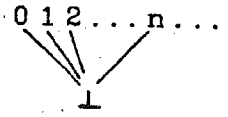
Together with the results about strong equivalence, the sequentiality and stability theorems are also fundamental in the study of optimal computations, see [7, 14, 13, 28].

4. Continuous interpretations and models.

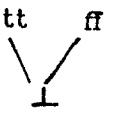
We define least fixpoint models, show their approximation continuity and define fully abstract models. Again we establish a strong distinction between the λ-calculus and the first-order calculus: models concern the first one, while first order interpretations are defined for the second one.

4.1. Interpretations.

We assume a basic knowledge of complete partial orders or cpo's, i.e. partial orders having a least element $\perp$ and such that any directed set has a lub. With the two ground types of *PCF* we associate the following flat cpo's:



$I(\iota)=N_\perp$



$I(o)=T_\perp$

And we interpret any *PCF* constant f by the associated continuous function $I(f)$, so that one has for example $I(\pm 1)(\perp)=\perp$ and $I(\pm 1)(n)=n+1$. We then obtain the *standard interpretation* S of *PCF*.

Formally, an interpretation is a many-sorted continuous algebra where there is one cpo per ground type and one continuous function per basic function symbol, with the appropriate type. The notion of interpretation is of course definable for any set of ground types and any set of function symbols. Given an interpretation I and a term t of A, one defines as usual the *value* $I(t)$ of t in I by induction on the size of t, and for any infinite T in A^∞ one sets $I(T)=lub\{I(t)\,|\,t\leq_\eta T\}$.

One obtains a particular interpretation from A^∞ itself, considering each function symbol as a tree constructor. This interpretation is nothing but the *free continuous algebra* generated by the constants. It will always be called I^f.

The standard interpretation S of *PCF* relates to δ-rules as follows:

4.1.2. Proposition: Let $t\in A$ in *PCF*. One has $t \overset{\bullet}{\to}_\delta n$ iff $S(t)=n$.

proof: easy induction on the size of t.

Using interpretations is therefore a way of forgetting about δ-rules in our class of languages (indeed Milner [32] considers only interpretations and does not introduce δ-rules). In the same way as δ-rules, interpretations may then be used for defining orderings on arbitrary terms. The ordering $\leq_I$ associated with I will be called the *operational ordering* determined by I.

4.1.3. Definition: Let I be an interpretation. We set $M\leq_I M'$ if for any program context $P[\,]$ one has $I(BT(P[M]))\leq I(BT(P[M']))$.

For *PCF*, it is not obvious that the orderings $\leq_S$ and $\leq_{op}$ coincide. That fact

will follow from the next result, which shows an essential extensionality property of $\leq_I$. For arbitrary terms, the operational semantics is defined in terms of behaviour in program contexts $P[]$. Putting a term M in such a context is a rather complex operation: the context may bind free variables of M, pass arguments to λ's of M etc. The result shows that the full generality of the context operation is already obtained by simple argument passing.

4.1.4. Notation: Let M be a term with free variables in $x_1, x_2, \cdots, x_n$. We call a *closure* $\dot{M}$ of M the (closed) term $\lambda x_1 x_2 \cdots x_n.M$.

4.1.5. The Context Lemma: Let M, M' be terms of the same type having their free variables among $x_1, x_2, \cdots, x_n$. Let $\dot{M}, \dot{M}'$ be the terms $\lambda x_1 x_2 \cdots x_n.M$ and $\lambda x_1 x_2 \cdots x_n.M'$. Then $M \leq_I M'$ holds iff $\dot{M}\overline{N} \leq_I \dot{M}'\overline{N}$ and iff $I(BT(\dot{M}\overline{N})) \leq I(BT(\dot{M}'\overline{N}))$ for any vector $\overline{N}$ such that $\dot{M}\overline{N}$ and $\dot{M}'\overline{N}$ are programs.

proof: The original proof by Milner [32] is valid only for combinatory logic, and does not extend to λ-calculus. A complete proof, may be found in [7, 10]. The difficulty lies in the "if" direction; one shows that $I(t)=n$ implies $I(P[M'])=n$ for any $t \leq_0 BT(P[M])$. This is done by a rather complex lexicographic induction on the number of free variables of MM', the size of t, the length of the leftmost reduction of $P[M]$ to hnf, and the number of holes in $P[]$.

4.1.6. Corollary: If P and P' are programs, then $P \leq_I P'$ holds iff $I(BT(P)) \leq I(BT(P'))$.

This is enough for showing that the natural interpretation of *PCF* captures its operational semantics:

4.1.7. Corollary: The relations $\leq_{op}$ and $\leq_S$ coincide in *PCF*.

proof: By 4.1.2 and 4.1.6.

4.2. Least fixpoint models.

We use the model definition introduced in [7, 8, 10], which departs slightly from the classical ones, found for example in [3, 21, 30]. In usual definitions, one introduces a value domain D^σ at each type σ, and one interprets any term as a function from environments (functions from variables to values) to value domains. We shall be a little more abstract, and introduce another set E^σ in which terms will be directly interpreted, together with a mapping *eval* from $E \times ENV$ to D which will allow to compute values of terms in environments (defined as usual on D's). It may happen that E is still a set of functions from ENV to D, *eval* being then just function application, but this is not necessary. Even in that case the explicit introduction of E has advantages, especially for

manipulating orderings between functions which make sense only on a particular function class (say stable functions); then E will be just that class. In other models such as the sequential algorithm models, objects are not functions anymore, and the introduction of E's is somewhat necessary. See [8, 30, 26] for an extensive discussion of that point (including the relation between our models and classical lambda algebras).

4.2.1. Definition. A (continuous) *premodel* is given by

- A cpo $<E^\sigma, \leq_E^\sigma, \perp^\sigma>$ at each type σ, called the *semantic domain* of type σ.
- A cpo $<D^\sigma, \leq^\sigma, \perp^\sigma>$ at each type σ, called the *value domain* of type σ. The infinite product of the D^σ indexed by the variables x^σ is called the *environment domain* ENV, and its elements are written ρ, ρ', etc. We call $\rho(x)$ the x-component of ρ, and $\rho[x \leftarrow v]$ the environment ρ' such that $\rho'(x)=v$ and $\rho'(y)=\rho(y)$ for $y \neq x$.
- A continuous *evaluation mapping* $eval: E^\sigma \times ENV \to D^\sigma$ at each type σ. We abbreviate $eval(e,\rho)$ into $e\rho$.
- A continuous *application mapping* $app: D^{\sigma\to\tau} \times D^\sigma \to D^\tau$ for each σ and τ. We write dd' instead of $app(d,d')$.

A premodel is *environment order extensional* if $e\rho \leq e'\rho$ for all ρ always implies $e \leq_E e'$, is *environment extensional* if $e\rho = e'\rho$ for all ρ always implies $e=e'$, is *value order extensional* if $dd'' \leq d'd''$ for all d'' always implies $d \leq d'$, and is *value extensional* if $dd''=d'd''$ for all d'' always implies $d=d'$. A premodel which is both environment and value (order) extensional is said to be *(order) extensional*. Usually only environment order extensional premodels are considered in the literature, but we shall see that this is far from being sufficient.

4.2.2. Definition: Given an interpretation I, a *least fixpoint model* or simply *model* $\mathcal{M}$ of (PCF, I) is the pair of a premodel also called $\mathcal{M}$ and of a set of *semantic functions* $\mathcal{M}[\![\,]\!]: PCF^\sigma \to E^\sigma$ such that:

(i) $\mathcal{M}[\![f]\!]\rho d_1 d_2 \cdots d_n = I(f)(d_1, d_2, \cdots, d_n)$ for all constant f of arity n and all $\rho, d_1, d_2, \ldots, d_n$

(ii) $\mathcal{M}[\![\Omega]\!]\rho = \perp$ at all types for all ρ

(iii) $\mathcal{M}[\![x]\!]\rho = \rho(x)$ for all ρ, x

(iv) $\mathcal{M}[\![MN]\!]\rho = (\mathcal{M}[\![N]\!]\rho)(\mathcal{M}[\![N]\!]\rho)$ for all M,N,ρ

(v) $\mathcal{M}[\![\lambda x.M]\!]\rho d = \mathcal{M}[\![M]\!]\rho[x \leftarrow d]$ for all x,M,ρ,d

(vi) $\mathcal{M}[\![YM]\!]\rho = fix(\mathcal{M}[\![M]\!]\rho)$ for all M,ρ, where $fix(f) = lub\{f^n(\perp)\}$

(vii) $\mathcal{M}[\![M]\!] \leq_E \mathcal{M}[\![N]\!] \Rightarrow \forall C[].\ \mathcal{M}[\![C[M]]\!] \leq_E \mathcal{M}[\![C[N]]\!]$

(viii) If for all $X \subset PCF$, if the set $\{\mathcal{M}[\![M]\!] \mid M \in X\}$ is directed and has lub $\mathcal{M}[\![N]\!]$, then for any context $C[]$ the set $\{\mathcal{M}[\![C[M]]\!] \mid M \in X\}$ has lub $\mathcal{M}[\![C[N]]\!]$

(ix) If $\rho(x)=\rho'(x)$ for all x free in M, then $\mathcal{M}[\![M]\!]\rho=\mathcal{M}[\![M]\!]\rho'$

(x) If $M \overset{*}{\rightarrow} N$ then $\mathcal{M}[\![M]\!]=\mathcal{M}[\![N]\!]$

A model is a (η)-model if it satisfies one of the equivalent conditions:

(η_1) $\mathcal{M}[\![\lambda x.Mx]\!] = \mathcal{M}[\![M]\!]$ if x is not free in M.

(η_2) $\mathcal{M}[\![Mx]\!] \leq_E \mathcal{M}[\![Nx]\!]$, x not free in MN, implies $\mathcal{M}[\![M]\!] \leq_E \mathcal{M}[\![N]\!]$.

(η_2) $\mathcal{M}[\![Mx]\!] = \mathcal{M}[\![Nx]\!]$, x not free in MN, implies $\mathcal{M}[\![M]\!] = \mathcal{M}[\![M]\!]$.

The condition (i) expresses that the semantics of constants should agree with their interpretation; (ii)-(v) are classical; (vi) expresses that Y acts as a least fixpoint operator; (vii)-(viii) require the context operation (i.e. the formation laws of PCF) to be continuous, (ix) forbids some "crazy" models, and (x) requires semantics to be preserved by reductions. In environment extensional models (vii), (viii) and (x) follows from the other axioms, but this is not true in general. Clearly all extensional models are (η)-models, but the converse is false: a model which is neither environment nor value extensional may still be an (η)-model. This is for example the case for the sequential algorithm model [12].

Given a model $\mathcal{M}$, we set $M \leq_{\mathcal{M}} M'$ if $\mathcal{M}[\![M]\!] \leq_E \mathcal{M}[\![M']\!]$, which implies (but is not necessarily equivalent to) $\mathcal{M}[\![M]\!]\rho \leq \mathcal{M}[\![M']\!]\rho$ for all ρ.

Additional properties such as ω-algebraicity will be said to hold for a model iff they hold for each domain. (recall that a point x of a cpo D is *isolated* if for any directed $X \subset D$, $x \leq lub\ D$ implies $\exists y \in D.\ x \leq y$, and that a cpo is ω-algebraic if it has denumerably many isolated points and if for any x the set of isolated points below x is directed and has lub x). We shall use also Plotkin's notion of *SFP cpo*. Call *projection* any continuous $f : D \rightarrow D$ such that $f \circ f = f$ and $f(x) \leq x$ for all x, and say that a projection is *finite* if its range is. Then a cpo is SFP if its identity function is the pointwise lub of an increasing sequence of finite projections (equivalently, the SFP cpo's are obtained as ω-colimits of chains of

finite posets in the category of cpo's and embeddings).

4.3. Approximation continuity, operational adequation and full abstraction.

The next result expresses that every least fixpoint model is continuous w.r.t. approximations:

4.3.1. Theorem: Let $\mathcal{M}$ be a least fixpoint model. Then for any M one has $\mathcal{M}[\![M]\!] = lub\{\mathcal{M}[\![t]\!] \mid t \leq_\Omega BT(M)\}$.

proof: The difficulty is to show that the first member is not greater than the second one. Extend the semantic function to PCF^l by setting

$$\mathcal{M}[\![Y^n M]\!] = \mathcal{M}[\![\underbrace{(M(M(\ldots(M\,\Omega)\ldots))}_{n}]\!]$$

Using (viii), show $\mathcal{M}[\![YM]\!] = lub\{\mathcal{M}[\![Y^n M]\!] \mid n \in N\}$. Then given $n>0$ and M, let M^n be the term where all Y's receive label n. Let N be the normal form of N in PCF^l, which exists by the FTPL property, let $t=\omega(N)$. Check $\mathcal{M}[\![M^n]\!] \leq_E \mathcal{M}[\![t]\!]$. Complete proof in [7].

4.3.2. Corollary: Let I be an interpretation, let $\mathcal{M}$ be a model of I. Then for any program P and any ρ one has

$$\mathcal{M}[\![P]\!]\rho = lub\{\mathcal{M}[\![t]\!]\rho \mid t \leq_\Omega BT(P)\} = I(BT(P))$$

Hence the value of any program in any model of I depends only on I and is the same for every model. As a consequence, any model is adequate w.r.t. operational semantics:

4.3.3. Theorem: Let I be an interpretation, let $\mathcal{M}$ be a model of I. Then $M \leq_{\mathcal{M}} M'$ implies $M \leq_I M'$.

proof: Let P be a program. Then $\mathcal{M}[\![P]\!]\rho = I(BT(P))$ for any ρ by 4.3.1 and the definition of interpretation and models. Now $\mathcal{M}[\![M]\!] \leq_E \mathcal{M}[\![M']\!]$ implies $\mathcal{M}[\![P[M]]\!]\rho \leq \mathcal{M}[\![P[M']]\!]\rho$ for any ρ by the definition of models, and the results follows.

The converse property need not be true, as we shall see in the sequel. We set the following definition:

4.3.4. Definition: A model $\mathcal{M}$ of I is said to be *fully abstract* w.r.t. I if the orderings $\leq_{\mathcal{M}}$ and $\leq_I$ coincide.

5. Construction and characterization of fully abstract models.

As we said in the introduction, we do not know how to construct a fully abstract model of (PCF,S) by "semantic" ways. In this section, we recall Milner's syntactic construction of such a model, and his uniqueness result. We also study the case of the free interpretation.

5.1. Construction of a fully abstract model of (PCF,S).

A key to full abstraction is definability of isolated points of an ω-algebraic model:

5.1.1. Notation: Given a model $\mathcal{M}$ and a point d of D^σ, we say that d is *definable* if there exists a closed M such that $d=\mathcal{M}[\![M]\!]\perp$.

5.1.2. Lemma: Let $\mathcal{M}$ be an ω-algebraic extensional model of (PCF,S) such that any isolated point of any D^σ is definable. Then $\mathcal{M}$ is fully abstract w.r.t. S.

proof: Assume $M\leq_S M'$ for M,M' closed. One has $M\overline{N}\leq_S M'\overline{N}$ for all closed $\overline{N}$ such that $M\overline{N}$ is ground, hence, using 4.1.5, 4.1.7 and the definability hypothesis, $\mathcal{M}[\![M]\!]\rho\overline{d} \leq \mathcal{M}[\![M']\!]\rho\overline{d}$ for all ρ and all isolated $\overline{d}$, hence finally $\mathcal{M}[\![M]\!]\leq_E\mathcal{M}[\![M']\!]$ by extensionality.

Now the idea is to use the terms themselves to construct a model. For each type σ, consider the set of equivalence classes of closed terms modulo $=_S$ (or equivalently $=_{op}$), calling $|M|$ the class of M. This set is clearly a partial order, with the class of Ω as least element; it is not a cpo, although some directed sets may have lubs. To turn it into a cpo, one needs to use a completion process other than the classical ideal completions which would add new lubs to any directed sets. There exist standard "conservative completions" which do the job, see [32, 16, 7]. This determines a cpo D^σ at each type. Define application by setting $|M||N| = |MN|$, with extension by continuity to the whole domains. By the Context Lemma, $|M||N| \leq_S |M'||N|$ for all N implies $|M|\leq_S|M'|$, so that application is order-extensional. Construct then an environment order extensional model $\mathcal{S}$ by taking for E^σ the set of all continuous functions from ENV to D^σ ordered pointwise, the semantic function $\mathcal{S}[\![]\!]$ being defined in the only possible way.

5.1.3. Theorem: The model $\mathcal{S}$ is fully abstract w.r.t. S.

proof: The ground domains are clearly SFP, and moreover one may easily for them define an increasing chain Φ_n of terms denoting finite projections having the identity as lub, by projecting $N_\perp$ to the finite cpo containing only the integers from 0 to n. Then one can define such a chain of projections at any type

by a simple induction on types, so that each D^σ is SFP. The isolated points of D^σ are then the classes of the form $|\Phi_n M|$. Hence they are all definable. The result follows by the previous lemma.

The construction may be generalized to general SFP interpretations where ground finite projections are definable, see [32, 7].

5.2. Uniqueness of the fully abstract model of (PCF,S).

Milner [32] also shows that the fully abstract model just obtained is unique among extensional models.

5.2.1. Lemma: All extensional models of (PCF,S) are SFP and hence ω-algebraic.

proof: The finite projections of 5.1.3 exist in any model, and may furthermore be defined by an increasing chain of terms w.r.t. $\leq_\Omega$. Hence all domains are SFP.

5.2.2. Theorem: An extensional model of (PCF,S) is fully abstract if and only if it is order extensional and such that all the isolated points of the D^σ are definable.

proof: The "if" direction has already been shown in 5.1.3. For the "only if" direction, one chooses a non-definable finite point d_0 of a D^σ with σ of minimal size, and one constructs two closed terms M_1 and M_2 such that $\mathcal{M}[\![M_1]\!]\perp d = \mathcal{M}[\![M_2]\!]\perp d$ for all definable d, so that $M_1 =_S M_2$ by the context lemma, and such that $\mathcal{M}[\![M_1]\!]\perp d_0 \neq \mathcal{M}[\![M_2]\!]\perp d_0$, i.e. $M_1 \neq_{\mathcal{M}} M_2$. See [32, 7] for the exact construction.

5.2.3. Corollary: The value domains of any two extensional fully abstract models of (PCF,S) are isomorphic at any type.

proof: Given two fully abstract models $\mathcal{M}_1$ and $\mathcal{M}_2$, the isomorphism h at type σ simply associates $h(x)=\mathcal{M}_2[\![M]\!]$ with any isolated $x=\mathcal{M}_1[\![M]\!]$.

The results are valid for the class of *articulate* interpretations, see [32, 7]. Later on we shall freely refer to Milner's model as *the* fully abstract model of PCF, forgetting that there may still exist non-extensional fully abstract models.

5.3. The initial model.

In the same way as we have constructed a free (initial) interpretation, we can construct initial models. Define a *morphism* of models as a collection of continuous maps $\Theta_E: E \to E'$ and $\Theta_D: D \to D'$ such that for all d_1, d_2, e, ρ, M:

$$\Theta_D(d_1 d_2) = \Theta_D(d_1)\Theta_D(d_2)$$

$$\Theta_D(e\rho) = \Theta_E(e)\Theta_D(\rho)$$

$$\Theta(\mathcal{M}[\![M]\!]) = \mathcal{M}'[\![M]\!]$$

Define now the initial model I by taking for E the set of all Boehm trees, for D the set of all closed Boehm tree, for *eval* the subsitution operation extended to Boehm trees, for *app* the application operation defined by the context $[]_1[]_2$ as in 3.5.8.

5.3.1. Theorem: I is the initial model of the free interpretation I^f: for any model M of I^f there exists a unique morphism from I to M.

One may also construct an initial η-model I_η: noticing that the η-expansion rule $M \to \lambda x.Mx$, x not free in M, always terminates in a typed calculus such as PCF, one can take the E's (D's) as the sets of (closed) Boehm trees in maximal η-expansion. One has then to check that substitution and application preserve the property of being in maximal η-expansion, see [22].

5.3.2. Theorem: The model I_η is order extensional and is initial among (η)-models, extensional models and order extensional models of I^f. It is fully abstract w.r.t. I^f.

proof: The proof is rather technical, purely syntactic, and very different from Milner's proof for S. The main difficulty is to show that I_η is order extensional. See [7, 10].

The same results holds for any language with enough constants; it may be wrong if the set of constants is too small, see [7, 10].

Here fully abstract models are definitely not unique:

5.3.4. Proposition: Any (η)- or extensional model of I^f is fully abstract.

proof: full abstraction is clearly preserved by morphisms.

6. Categorical construction of models.

So far we have seen syntactic model constructions, where the objects of the models are obtained from the terms themselves. But of course the most interesting model constructions are the semantic ones, in which the objects are interesting on their own and do not depend on the language; usually they are functions, but as we shall see later on other objects than functions may be interesting. The natural framework in which to study semantic models is that of cartesian closed categories [7, 10, 29, 37]. Intuitively a category is cartesian closed if the set of arrows from D to E can always be represented by an object $(D \to E)$.

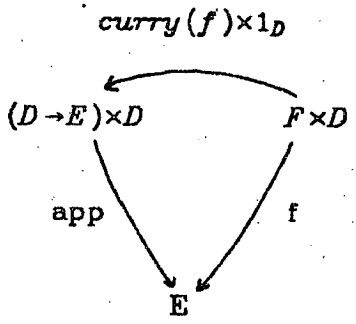
6.1. Λ-categories.

6.1.1. Definition: In a category $\mathbb{C}$, we call $\mathbb{C}(D,E)$ the hom-set (set of arrows) from D to E. A category $\mathbb{C}$ is *cartesian closed* or is a ccc if it has the following properties:

(i) It is product-closed, i.e. contains all finite products. Hence it contains a terminal object T (the product of no objects). The pair of $f:D\to E$ and $g:D\to F$ is written $\langle f,g\rangle:D\to E\times F$, and the product of $f:D\to D'$ and $g:E\to E'$ is written $f\times g:D\times D'\to E\times E'$. The i-th projection is called π_i.

(ii) For any objects D,E, there exists an object $(D\to E)$ and an arrow $app:(D\to E)\times D\to E$ such that for any $f: F\times D\to E$ there exists a unique arrow $curry(f): F\to(D\to E)$ satisfying

$$app\circ(curry(f)\times 1_D) = f$$



In a cartesian closed category $\mathbb{C}$, we set $\overline{D}=\mathbb{C}(\mathsf{T},D)$. The sets $\mathbb{C}(\mathsf{T},(D\to E))$ and $\mathbb{C}(D,E)$ are always isomorphic.

Given two arrows $f:D\to(E\to F)$ and $g:D\to E$, we set $Sfg = app\circ\langle f,g\rangle$ (in categories of sets and functions, one has $Sabc=ac(bc)$.

6.1.2. Definition: A *cpo-enriched* category [39, 44] is a category such that each $\mathbb{C}(D,E)$ is turned into a cpo $\langle\mathbb{C}(D,E),\leqq,\perp\rangle$ in such a way that composition is continuous.

6.1.3. Definition: A Λ-category is a cpo enriched cartesian closed category such that

(i) The composition is left-strict, i.e. satisfies $\perp\circ f = \perp$ for all f.

(ii) The pairing operation $\langle -,-\rangle$ (or equivalently the product functor $\times$) is order-continuous and strict (i.e. $\langle\perp,\perp\rangle=\perp$)

(iii) The *curry* function is an order-isomorphism from $\mathbb{C}(D\times E,F)$ to $\mathbb{C}(D,(E\to F))$. The S operator is left-strict, i.e. such that $S\perp f=f$ for all f.

6.2. Model construction from Λ-categories.

Although it is not necessary, we shall add an hypothesis which makes life simpler: we shall consider only Λ-categories which are closed by denumerable products. It will allow us to turn the set of environments into an object (otherwise one has to count variables and to introduce finite environments relative to a finite set of variables, see [26]).

6.2.1. Definition: Let I be an interpretation, let $\mathbb{C}$ be a Λ-category. We construct a model $\mathbb{C}$ of I in the following way:

- For each ground type κ, choose an object C^κ such that the cpo's $\overline{C^\kappa}$ and $D\kappa$ are isomorphic;

- For each constant $f^{\kappa_1\times\kappa_2\times\cdots\times\kappa_n\to\kappa}$ choose an arrow $\overline{f}:C^{\kappa_1}\times C^{\kappa_2}\times\cdots\times C^{\kappa_n}\to C^\kappa$ such that $S(\ldots(S(S\overline{f}d_1)d_2)\ldots)d_n=d$ iff $I(f)(d_1,d_2,\ldots,d_n)=d$.

- For each type $\sigma\to\tau$, set $C^{\sigma\to\tau}=(C^\sigma\to C^\tau)$; set $D^\sigma=\overline{C^\sigma}$, and for f in $D^{\sigma\to\tau}$, d in D^σ, set $app(f,d)=Sfd$;

- Define the object Env as the infinite product of the C^σ's indexed by the variables. Then $ENV=\overline{Env}$ is the environment over the D^σ's. Given a variable x^σ, let π_x be the x-th projection, let S_x be the unique arrow from $Env\times D^\sigma$ to Env such that $\pi_x\circ S_x=\pi_2$ and $\pi_y\circ S_x=\pi_y\circ\pi_1$ for $y\neq x$. Then $\pi_x\circ\rho=\rho(x)$ and $S_x\circ\langle\rho,d\rangle=\rho[x\leftarrow d]$ for all ρ,d.

- Set $E^\sigma=\mathbb{C}(Env,C^\sigma)$; for e in E^σ and ρ in ENV, set $eval(e,\rho)=e\circ\rho$.

- Define the semantic function $\mathbb{C}[\![\,]\!]$ by:

(i) $\mathbb{C}[\![f]\!]=\overline{f}$ for all constant f

(ii) $\mathbb{C}[\![\Omega]\!]=\perp$ at all types

(iii) $\mathbb{C}[\![x]\!]=\pi_x$ for all variable x

(iv) $\mathbb{C}[\![MN]\!]=S\ \mathbb{C}[\![M]\!]\ \mathbb{C}[\![N]\!]$ for all M,N

(v) $\mathbb{C}[\![\lambda x.M]\!]=curry(\mathbb{C}[\![M]\!]\circ S_x)$ for all M,x

(vi) $\mathbb{C}[\![YM]\!]=lub\{S(\ldots(S(S\,\mathbb{C}[\![M]\!]\Omega)\Omega)\ldots)\Omega\}$ for all M of type $\sigma\to\sigma$

The above definitions are nothing but the "abstract" version of the equations of the model definition: environment and value variables have been removed in an usual categorical style.

6.2.2. **Theorem:** $\mathbb{C}$ is a least fixpoint η-model of I.

proof: see [7, 10]. The main difficulty is to show the validity of β-conversion, since we cannot use extensionality as in function models. Extensionality has to be replaced by uniqueness of $curry(f)$ for all f.

6.2.3. **Remark:** An important fact is that (η) is not equivalent to extensionality (concreteness) of $\mathbb{C}$, but is *always* valid, even if the arrows of $\mathbb{C}$ are not functions. This is due to the introduction of E in our model definition, see [8, 10]. That fact is absolutely essential in the study of sequential algorithms, see [12] and section 9. One can also construct models which do not satisfy (η) by taking $C^{\sigma\to\tau}$ as being a *retract* of $(C^\sigma \to C^\tau)$, see [37].

7. Continuous function models.

7.1. The continuous function model.

The simplest Λ-category is of course that of cpo's and continuous functions. The exponentiation of two cpo's D and E is the set $[D\to E]$ of all continuous functions from D to E ordered by the *extensional* of *pointwise* ordering $\leq_\bullet$, defined by $f \leq_\bullet g$ iff $f(x)\leq g(x)$ for all x. The hom-sets are ordered by the same ordering. The model obtained is clearly order extensional. The continuous function models was originally introduced in [31]. Plotkin [34] showed the following result:

7.1.1. **Proposition:** The continuous function model is not fully abstract.

proof: Let *por* be the continuous function from boolean pairs to booleans defined by $por\ tt\ \bot = por\ \bot\ tt = tt$ and $por\ ff\ ff = ff$. By the stability theorem, the function *por* is not *PCF*-definable, and the result follows by 5.2.2.

However Plotkin obtained a very nice full abstraction result by extending the language, adding not exactly the "parallel or" but a similar "parallel if" construct:

7.1.2. **Notation:** Let *PCFP* be the *PCF* language augmented with two constants $\underline{pif}_\iota$ and $\underline{pif}_o$ having the following rules:

$\underline{pif}\ tt\ x\ y \to x$

$\underline{pif}\ ff\ x\ y \to y$

pif $M\ c\ c \to c$ for any constant c and term M.

In the standard interpretation of *PCFP*, the *pif*'s are interpreted as in the rules.

7.1.3. Theorem: The continuous function model is fully abstract w.r.t. the standard interpretation of *PCFP*.

proof: show that all finite points of the model are definable, see [34], and use 5.1.2.

However many important properties of *PCF* are lost, such as stability and sequentiality (consider the term *pif* $\Omega\ \Omega\ \Omega$, which has no sequentiality index). Indeed the computations in *PCFP* have to be "parallel". For assume we want to compute a term *por* $M_1\ M_2$ (*por* is easy to define from *pif*); we then have to compute the values of M_1 and M_2 in parallel: if we start computing M_1 alone, it may happen that we never end, while computing M_2 would have given the value *tt*, and symmetrically. This makes *PCFP* not very interesting as a language, since it requires a parallel evaluator but has really no parallel programming primitives.

7.2. Full subcategories of continuous functions.

The continuous model construction for a language such as *PCF* starts from two very simple "flat" partial orders, and proceeds by taking exponentiations. Then one is far from generating arbitrarily general cpo's, and it is interesting to look for additional properties of the actually generated stuctures. The right way to do it is of course to look for conditions such that the cpo's which satisfy them form cartesian closed full subcategories of the category of cpo's, in other words to look for conditions which are preserved by products and exponentiation. If satisfied by the ground domains, they will be satisfied by all domains. Let us enumerate some classical properties:

- A cpo is *consistently complete* if any upperbounded subset has a lub, or equivalently if any nonempty subset has a glb. Consistent completeness is preserved by product and exponentiation.
- Say that a subset is *pairwise consistent* if any two points in the set have an upperbound. Then a partial order is *coherent* if any pairwise consistent set as a lub. Any coherent partial order is of course a consistently complete cpo. Coherence is preserved by product and exponentiation.

- The ω-algebraicity property is preserved by product, but *not* by exponentiation; Smyth [38] has shown that the largest cartesian closed subcategory of ω-algebraic is the category of SFP cpo's.
- The ω-algebraicity and consistent completeness (or coherence) properties are preserved together. The category obtained is a full subcategory of SFP.

Since the ground domains are coherent and ω-algebraic, all the domains of the continuous model of PCF have the same properties.

7.3. Event structures.

The category of ω-algebraic consistently complete cpo's is probably the most often considered in the literature. However there are many more interesting categories, where representation theorem exist which link "abstract" objects such as cpo's and "concrete" objects such as the concrete data structures of [25] or the event structures of [33, 46]. The theories of such "concrete" objects turns out to be crucial in the study of full abstraction, as we shall see later on. Here we shall concentrate on the simple theory of *continuous event structures* as studied by Winskel [46]. The idea, inherited from [25], is to introduce a notion of "information quantum" in Scott's theory of cpo's, any point of a cpo being nothing but the set of its quanta. A quantum is actually called an "event", as in the Petri Net Theory [33].

7.3.1. Definition: An *event structure* is a triple $\mathbb{E}=(E,\leq,\#)$ where E is a set of *events*, $\leq$ is a partial order on E called the *causality* order, and $\#$ is a *conflict* relation on E. A *state* of $\mathbb{E}$ is a subset x of E such that:

(i) x is *left closed*, i.e. $e\in x$ and $e'\leq e$ imply $e'\in x$. Intuitively, events may occur only after their causes.

(ii) x is *conflict-free*, i.e. such that $e,e'\in x$ implies *not* $e\#e'$. Hence two events in conflict may not occur together.

The states are ordered by inclusion, and are easily seen to form an ω-algebraic coherent cpo. But the obtained cpo's are more that just ω-algebraic. They are characterized in the following way:

7.3.2. Definition: A *prime* in a consistently complete poset is a point x such that for all $X\subset D$ upperbounded, $x\leq\forall X$ implies $\exists y\in X.\ x\leq y$. A consistently complete cpo is *prime algebraic* if every point is the lub of the set of primes it dominates. (Notice that any prime is isolated, and that any prime algebraic cpo is ω-algebraic.)

7.3.3. Theorem: The cpo of states generated by an event structure is prime algebraic and coherent. Conversely any prime algebraic coherent cpo is generated by some event structure.

proof: The primes are just the ideals generated by the events, i.e. the sets $\{e' \leqq e\}$ for fixed e's. Conversely, the primes of a prime algebraic cpo's may be considered as events; the causality relation is obtained from the ordering, and the conflict relation from the order incompatibility relation. See [46].

Prime algebraicity and coherence imply many other properties, such as distributivity

(D) $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ for all x and all upperbounded y, z

or Kahn-Plotkin axioms C and R, see [46].

The most interesting fact is of course the following one:

7.3.4. Theorem: The category of prime algebraic coherent cpo's and continuous functions is cartesian closed.

proof: The primes of $[D \to D']$ are the functions $(x \Rightarrow p')$ defined by $(x \Rightarrow p')(y) = p'$ if $x \leqq y$ and $(x \Rightarrow p')(y) = \bot$ otherwise, for x isolated and p' prime.

Hence all the domains of the continuous model of (PCF, S) are coherent and prime algebraic. In the study of event structures, one often uses the following additional property:

(F) Any isolated point dominates finitely many points.

Then property (F) is satisfied if any event is caused by only finitely many other events. Property (F) is *not* preserved by exponentiation: it is easy to see that the constant $\top$ function from $N_\bot$ to $\{\bot, \top\}$ is isolated and dominates infinitely many isolated functions, namely all those functions which have value $\top$ on a finite number of integers and $\bot$ elsewhere. The property (F) expresses that isolated elements are really "finite" as far as information is concerned. This is certainly a desirable property of "concrete" objects. That function spaces do not satisfy (F) shows that they are not "concrete". This may seem natural at first glance, but is in fact due to the nature of $\leqq_s$. In the next sections we shall see that in other categories function spaces can be really made concrete by using other orderings for functions.

8. Stable function models.

Our problem is now to construct Λ-categories "smaller" than that of continuous functions, where non-definable objects such as *por* or *pif* do not exist. To see what additional properties we need, we have to remember the syntactic properties of *PCF*. Precisely we used the stability and sequentiality properties for showing non-definability results. Hence these properties should be reflected in the semantics, and should also serve as links between the syntax and the semantics. With respect to stability, we should for example require the following property:

(ST) For all $e \leq \mathcal{M}[\![M]\!]$ there exists a least $M' \leq_0 M$ such that $e \leq \mathcal{M}[\![M']\!]$.

The property is not satisfied by the continuous functions model, as easily seen by taking $M = x\ tt\ tt$ and $e(\rho)=tt$ iff $\rho(x)=por$. Since sequentiality was stonger than stability, we should also require a stronger sequentiality property. But there is a difficulty: sequentiality was expressed in terms of *occurrences*, and we have for the moment no simple notion of "occurrence" in cpo's. We postpone the discussion of sequentiality to the next section, and discuss stability first.

8.1. Stable functions and the stable ordering.

The stability property is definable for arbitrary functions:

8.1.1. Definition: Let D, D' be cpo's. A continuous function $f: D \to D'$ is *stable* if it satisfies

For all x and $x' \leq f(x)$, there exists $M(f,x,x') \in D$ such that for all $z \leq x$, $x' \leq f(z)$ holds iff $M(f,x,x') \leq z$.

The set of stable functions from D to D' is called $[D \to_s D']$.

Clearly $M(f,x,x')$ is the least $z \leq x$ such that $x' \leq f(z)$. Of course the function *por* is not stable. The constant function, the identity function and the product projections are stable. All *PCF* constants are stable. The *BT* function is stable. The theory of stable functions is presented in [6, 7, 15] and we just indicate the main results.

8.1.2. Lemma: The composition of two stable functions is stable with $M(f \circ f, x, x'') = M(f, x, M(f', f(x), x''))$.

Hence the stable functions form a product-closed category. The next problem is to order stable functions. A first remark is that the extensional ordering $\leq_e$ is *not* the right ordering, since the application function

$app:[D\to_s D']\times D\to D'$ is not stable in general. To see that fact, let $D=\{\bot,\top\}$. Then $[D\to_s D]$ contains three functions: the constants $\bot$ and $\top$ and the identity 1. The point $M(app,(\top,\top),\top)$ does not exist, since one has $\top(\bot)=\top$, $1(\top)=\top$, but $1(\bot)=\bot$. Therefore the stable exponentiation of two cpo's cannot be ordered by $\leq_s$. The right ordering compares not only the values of the functions but also the way they are computed, i.e. their minimal points $M(f,x,x')$.

8.1.3. Definition: The *stable ordering* $\leq_s$ between stable functions is defined by

$f\leq_s g$ iff $f\leq_s g$ and $\forall x\in D,\ \forall x'\leq f(x),\ M(f,x,x')=M(g,x,x')$.

Coming back to the previous example, we see that $1\leq_s\top$ does *not* hold, since $M(1,\top,\top)=\top$ while $M(\top,\top,\top)=\bot$. The functions 1 and $\top$ do not use the same information in their argument for outputting their results.

We need some additional properties of domains:

8.1.4. Definition: A *stable cpo* is a consistently complete distributive cpo such that the glb function is continuous.

8.1.5. Theorem: Let D, D', be stable cpo's. Then $<[D\to_s D'],\leq_s,\bot>$ is a stable cpo and the application function $app:[D\to_s D']\times D\to D'$ is stable. The function *curry* is an order isomorphism from $[D\times D'\to_s D'']$ to $[D\to_s[D'\to_s D'']]$.

proof: The lub of any directed set of $[D\to_s D']$, the lub of two upperbounded point are always taken pointwise. The glb of two points is not pointwise unless they are compatible w.r.t. $\leq_s$. In general it is defined by

$$f\wedge g(x)=lub\{x'\leq f(x)\wedge g(x)\mid \forall y'\leq x'.\ M(f,x,y')=M(g,x,y')\}$$

Distributivity is crucial for showing these properties. For application, one checks

$M(app,(f,x),x')=(g,y)$ with $y=M(f,x,x')$ and $g(z)=x'\wedge f(y\wedge z)$.

8.1.6. Corollary: The category of stable cpo's and stable functions order-enriched by $\leq_s$ is a Λ-category.

Hence we have obtained a new model of *PCF*, which is extensional but not order extensional. It is not fully abstract.

8.1.7. Proposition: The stable model satisfies property (ST).

proof: easy induction on the size of terms.

To finish, let us mention that Property (F) *is preserved* with stable functions. In some sense that we shall make precise in a moment the ordering $\leq_s$ is more concrete than $\leq_e$.

8.1.8. Definition: A *SF-domain* is an ω-algebraic stable cpo satisfying property (F). (SF-domains were called dI-domains in [7]).

8.1.9. Proposition: The stable exponentiation of SF-domains is a SF-domain. The category of SF-domains and stable functions is cartesian closed.

8.2. Conditionally multiplicative functions.

Stability is a fairly natural condition. But it is technically often difficult to use, and we shall study here a slightly weaker condition of the same spirit, which turns out to be equivalent for SF-domains.

8.2.1. Definition: In a cpo D, write $x \uparrow y$ if x and y are upperbounded. Let D, D' be two stable cpo's. Then $f : D \to D'$ is said to be *conditionally multiplicative* or *cm* if it satisfies

$$\forall x,y,\ x \uparrow y \Rightarrow f(x \wedge y) = f(x) \wedge f(y)$$

8.2.2. Lemma: Any stable function is cm. If D is a SF-domain, then a function is stable iff it is cm.

Again the ordering $\leq_e$ is not adequate for cm functions: with the same example as before, one has $(1,\top) \uparrow (\top,\perp)$ but $app(1,\perp) \neq app(1,\top) \wedge app(\top,\perp)$. The right ordering is defined as follows:

8.2.3. Definition: Let $f, g : D \to D'$ be cm. We set $f \leq_{cm} g$ if $f \leq_e g$ and

$$\forall x,y,\ x \uparrow y \Rightarrow f(x) \wedge g(y) = f(y) \wedge g(x)$$

The definition is a bit surprising, but it obviously makes application cm. A composition of cm functions is easily seen to be cm. Finally one obtains:

8.2.4. Theorem: The category of stable domains and cm functions order enriched by $\leq_{cm}$ is a Λ-category. The associated model satisfies (ST).

proof: see [7, 15].

On SF-domains we have:

8.2.5. Proposition: Let D, D' be SF-domains, let $f, g : D \to D'$ be stable (i.e. cm). Then one has $f \leq_s g$ iff $f \leq_{cm} g$.

Therefore stable and cm functions are really identical on SF-domains.

8.3. Biordered stable models.

So far we have restricted the set of functions but we have lost order extensionality. Hence we have some strange functions in our model, such as functions which return *tt* for the identity and *ff* for the $\top$ constant on $\{\bot, \top\}$! Such functions cannot be defined in *PCF* (but they can indeed be defined in *CDS*, see section 9 and [12, 11]). The next step is to "intersect" the continuous and stable or cm models by manipulating *together* the orderings $\leq_s$ and $\leq_e$. The simplest construction uses cm functions.

8.3.1. Definition: a *bicpo* is a structure $<D, \leq_{cm}, \leq_e, ,>$ where:

(i) $<D, \leq_{cm}, \bot>$ is a cpo

(ii) $<D, \leq_e, \bot>$ is a cpo

(iii) The identity function 1: $<D, \leq_{cm}, \bot> \to <D, \leq_e, \bot>$ is continuous

(iv) The function $\wedge : D \times D \to D$ is the glb function on $<D, \leq_e, \bot>$ and is both $\leq_e$- and $\leq_{cm}$-continuous

(v) For any $X, X' \subset D$ directed, if for all $x \in X$ and $x' \in X'$ there exist $y \in X$, $y' \in X'$ such that $x \leq_e y$, $x' \leq_e y'$ and $y \leq_{cm} y'$, then $lub\ X \leq_{cm} lub\ X'$

Since $\wedge$ is $\leq_{cm}$-continuous, the $\leq_{cm}$-glb of two $\leq_{cm}$-upperbounded points x and y is simply $x \wedge y$. A function $f : D \to D'$ between two bidomains is said to be *cm* iff it is $\leq_e$-continuous and $\leq_{cm}$-cm.

We said nothing about $\leq_{cm}$-lubs, which may not exist. We say that a bicpo is *distributive* if the cpo $<D, \leq_{cm}, \bot>$ is distributive and if moreover the $\leq_{cm}$-lub of any two $\leq_{cm}$-upperbounded points is also their $\leq_e$-lub.

8.3.2. Theorem: The category of bicpo's (resp. distributive bicpo's) and cm functions is cartesian closed.

Now we obtain *two* models with the same carrier, according to the ordering used for enriching the category. Both satisfy (ST). The model ordered by $\leq_e$ is the most interesting one; it is "closer" to the fully abstract model than the continuous one, since it contains less objects. But it is still *not* fully abstract, since it contains some non definable objects, such as the three argument boolean function defined by $f(tt, ff, \bot) = tt$, $f(\bot, tt, ff) = tt$, $f(ff, \bot, tt) = tt$ which was shown non-definable in 3.6. The model ordered by $\leq_{cm}$ is less interesting at first glance since it is extensional but not order-extensional. However remember that any model is approximation continuous: then we have $\mathcal{M}[\![M]\!] \leq_{cm} \mathcal{M}[\![N]\!]$ if $BT(M) \leq_\Omega BT(N)$, which is stronger than simply having $\mathcal{M}[\![M]\!] \leq_e \mathcal{M}[\![N]\!]$. Hence

$\leq_{cm}$ acts as an "image" of the approximation ordering defined by Boehm trees, in the same way as $\leq_e$ acts as an image of the operational ordering $\leq_{op}$.

For stable functions the construction is more complex, and one uses the fact that stable functions and cm functions are the same on SF-domains. Hence one looks for a subcategory of bicpo's and stable functions with more restricted objects. The technique used is the same as for SFP: construct the limit of finite structures.

8.3.3. Definition: A *finite projection* on a bicpo D is a cm function $h: D \to D$ such that $h \circ h = h$ and $h(x) \leq_{cm} x$ for all x. A distributive bicpo is a *bidomain* if it admits an $\leq_{cm}$-increasing sequence of finite projections with limit the identity.

8.3.4. Proposition: If $<D, \leq_{cm}, \leq_e, \perp>$ is a bidomain, then $<D, \leq_{cm}, \perp>$ is a SF-domain.

proof: see [7].

As a corollary, a function between bidomains is cm iff if is $\leq_e$-continuous and $\leq_{cm}$ stable. Then we prefer to call it stable.

8.3.5. Theorem: the category of bidomains and stable functions is cartesian closed.

The definition of bidomains is not very elegant. A very nice characterization of bidomains has been obtained by Winskel [46], who showed that bidomains may be represented concretely by *stable event structures*. The key idea is to introduce both orderings at the level of events, decomposing the extensional ordering $\leq_e$ into two parts: the ordering $\leq_s$ and its "complement". The construction is a bit complex, but it gives some deep insight into the nature of the orderings. We believe that any further attempt towards constructing the fully abstract model should start from it.

8.4. Stability of the fully abstract model.

For the moment, the orderings $\leq_s$ and $\leq_{cm}$ seem to be only technical tools necessary for carrying out the constructions. It is time to give them a natural statusby exhibiting their syntactic properties. A first result shows that Milner's fully abstract model $\mathcal{S}$ is indeed a model of bicpo's, the ordering $\leq_{cm}$ being "hidden" in the structure.

8.4.1. Definition Let $\mathcal{S}$ be the fully abstract model of *PCF*. Define an ordering $\leq^{\sigma}_{cm}$ at each type σ by induction on the size of σ:

(i) $\leq_{cm}^{\sigma} = \leq_{op}$ if σ is ground;

(ii) $|M| \leq_{cm}^{\sigma\to\tau} |N|$ iff $|M| \leq_{op} |N|$ and $|MP|_\wedge|NQ| = |MQ|_\wedge|NP|$ if $|P|$ and $|Q|$ are $\leq_{cm}$-upperbounded (the glb function is easily defined by induction on types).

8.4.2. Theorem: The structures $<D_{\mathcal{S}}, \leq_{cm}, \leq_e, \perp>$ of the fully abstract model $\mathcal{S}$ are bicpo's. Moreover each $D_{\mathcal{S}}^{\sigma\to\tau}$ is included (up to isomorphism) in the set of cm functions from $D_{\mathcal{S}}^{\sigma}$ to $D_{\mathcal{S}}^{\tau}$, the ordering $\leq_{cm}$ defined on term classes being the restriction of the ordering $\leq_{cm}$ on functions.

proof: By induction on types, see [7].

One can also construct a model from the $D_{\mathcal{S}}$'s ordered by the ordering $\leq_{cm}$, and both orderings yield models satisfying (ST).

Unfortunately we are not able to show that the domains of the fully abstract model are bidomains, although we definitely believe it; the problem is to show that the $\leq_{cm}$-lubs are taken pointwise.

For explaining the role of $\leq_{cm}$, we emit the following conjecture:

8.4.3. Conjecture: Let d, d' isolated in $D_{\mathcal{S}}^{\sigma}$. Then $d \leq_{cm} d'$ holds iff there exist two closed terms M, M' such that $\mathcal{S}[\![M]\!]\perp = d$, $\mathcal{S}[\![M']\!]\perp = d'$ and $BT(M) \leq_0 BT(M')$.

The conjectured fact is easily seen to hold at first order, and we have many indications that it holds everywhere. Then $\leq_{cm}$ would be the exact semantic image of the syntactic ordering induced by Boehm trees in the same way as $\leq_e$ is exact image of $\leq_{op}$. Hence we could have in a single model the results obtained in the type free case by Wadsworth for D^∞ [42, 43] and by Barendregt and Longo for T^ω [2] in the type free lambda calculus. (We suspect that one can also obtain both results together in the type free case by solving $D = (D \to D)$ in bidomains).

9. Sequential models.

9.1. Sequential functions.

The next step is to construct sequential models. As we already said, a problem is to define a suitable notion of occurrence or *place* in cpo's, and to define a sequential function as a function for which there is a relation between increase of information in output and input places, as for BT in the syntactic sequentiality theorem 3.6.2. Several definitions [32, 36, 41] use the rank of a function argument in the argument list as the place notion. They lead to

interesting results: optimality properties [41, 13] or Sazonov's characterization of sequentially computable functionals in Scott's D^ω model when the construction starts from a flat domain [36]. However counting arguments is not the right way of thinking in product closed categories, where a function should be viewed indifferently as a two argument function or as a one argument function from a product. Hence the definition is not general enough for our purpose.

The best known definition was introduced by Kahn and Plotkin in their study of concrete data structures [25]. Since all details are given in another paper in this volume [11], we give only some indication here, in terms of event structures. The idea is to introduce *places* for events and to derive the conflict relation from the constraint that in any state a place can only hold one event. A well-known example is a record with variant, where the places are just the fields. The discriminant field can contain only one value, and according to which value it contains it gives access to other fields (and this defines the causality relation, also called the enabling or accessibility relation in that framework). Trees give another example: places are occurrences, and filling a place gives access to its sons. Sequential functions may then be defined as follows:

> A function f is sequential if for any input x, for any place p' accessible from $f(x)$, either p' is not filled in $f(y)$ for all $y \geq x$ or there exists a place p accessible from x such that p' filled in $f(y)$ implies p filled in y for all $y \geq x$.

The BT function of section 3 is sequential in that sense. All sequential functions are stable (this was not true with the other definitions mentioned).

A disappointing result was obtained in [12]:

9.1.1. Theorem: The category of concrete domains and sequential functions is not cartesian closed.

proof: Show that the only possible ordering for exponentiation is the stable ordering, and that sequential functions ordered that way do not form concrete data structures.

It is certainly possible to define sequential functions in more general structures than concrete data structures, but we suspect that the same negative result will always hold.

9.2. Sequential algorithms.

For building sequential models we have therefore to abandon functions. A solution is to introduce *sequential algorithms* as arrows [11, 12, 17], where the computation strategy is part of the object as well as the input-output relation. These objects are studied elsewhere in this volume [11]. They can be viewed "concretely" as program texts ordered by the Ω-match ordering $\leq_\Omega$, or "abstractly" as function - computation strategy pairs ordered using the stable ordering. In some sense their operational and denotational semantics are unified.

Sequential algorithms form a Λ category. The model obtained has the property that the function $\mathcal{M}[\![\,]\!]$ itself is sequential and may be turned into an algorithm. The model is not extensional (although it satisfies (η)) and not fully abstract.

But another programming language was developped from sequential algorithms (and implemented): the language CDS, for which the algorithm semantics is fully abstract. The point is that the operational semantics changes: in *PCF* only programs have observable values, so that the operational semantics of terms can be defined only by placing them in program contexts. On the contrary, in CDS *any* term has an observable semantics, since higher order objects are no less concrete than ground types ones. Full abstraction comes from the fact that any finite algorithm at any type is definable. See [11, 19] for details.

9.3. Towards the fully abstract model.

We are currently trying to do with sequential algorithms what we have already done with stable functions: keep the extensional ordering $\leq_e$ in the construction. A clue is to use Winskel's ideas of multi-ordered event structures, see 8.4 and [46]. The events should be ordered by both the "algorithm" and the extensional ordering, the stable ordering being obtained for free as the quotient of the algorithm ordering by the extensional equality. See [18] for such an attempt. It might be the case that we indeed get the fully abstract model that way. Of course definability of isolated elements has to be shown. This has no chance of being trivial, as shown by an example: assume we want to define a functional F of type $((o\times o\to o)\to o)$ such that $F(f)=tt$ iff $f\perp\perp = \perp$ and either $f\ tt\perp = tt$ or $f\perp tt = tt$. Let $g=\lambda x.\ cond\ x\ tt\ \Omega$; then F is easily defined using the parallel or *por* as the semantics of the following term:

$$\lambda f.\ g(\underline{par}(f\ \underline{tt}\ \Omega)(f\ \Omega\ \underline{tt}))$$

At first glance it seems that F is not sequential and not definable without $\underline{par}$. However F is indeed sequential and PCF-definable, as the semantics of the following term:

$$\lambda f.g(f\ (g(f\ \underline{tt}\ \Omega))\ (g(f\ \Omega\ \underline{tt})))$$

The imbrication of f's is crucial. One may construct examples where an arbitrary number of internal calls of the same function is necessary, and we do not know how to treat this phenomenon in general.

10. Conclusion.

We left the story unfinished, since the original problem is still unsolved. We keep working on it, but we also believe that the side effects we mentioned are as interesting as the problem itself. In particular the experience of *CDS* has shown that other notion of operational observation make sense, not limited to observation of ground type terms [11]. Other extensions of the present work are interesting:

- Extension to languages with richer type structure, in particular with recursively defined types. Most of the techniques used here should apply directly. The syntactic results are the same, and for semantic model construction one has just to require domain equations to be solvable in the category, which raises usually no problem [11, 39, 37].
- Extension to languages which richer ground types and abstract data types. The work of [23] is very encouraging.
- Extension of the notion of sequentiality, which should not be bound to concrete data structures. Some axiomatic definition should work in a much more general framework. One should indeed *define* what is a sequential language.

Acknowledgements:

We thank G. Kahn, M. Nivat, G. Plotkin and G. Winskel for uncountably many helpful discussions.

References

1. E. Astesiano and G. Costa, "Non-determinism and Fully Abstract Models," *RAIRO Informatique Theorique* 14(4), pp.323-347 (1980).

2. H. Barendregt and G. Longo, "Equality of Lambda Tems in the Model T^{ω}," pp. 303-338 in *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, ed. J.P Seldin and J.R. Hindley, Academic Press (1980).

3. H.P. Barendregt, *The Lambda Calculus - its Syntax and Semantics*, North Holland Pub. Co. (1981).

4. G. Berry, "Bottom-up Computations of Recursive Programs," *RAIRO Informatique Théorique* 10(3), pp.47-82 (1976).

5. G. Berry, "Séquentialité de l'Evaluation Formelle des Lambda-expressions," pp. 67-80 in *Program Transformations, Proc 3rd Int. Coll. on Programming*, ed. B. Robinet, DUNOD, Paris (1978).

6. G. Berry, "Stable Models of Typed Lambda-calculi ," pp. 72-89 in *Proc. 5th Coll. on Automata, Languages and Programming*, Lectures Notes in Computer Science 62, Springer-Verlag (July 1978).

7. G. Berry, "Modèles Complètement Adéquats et Stables des Lambda-calculs Typés," Thèse de Doctorat d'Etat, Université Paris VII (D).

8. G. Berry, "On the Definition of Lambda-calculus Models," pp. 218-230 in *Proc. Int. Coll. on Formalization of Programming Concepts*, Lecture Notes in Computer Science 107, Springer-Verlag (1981).

9. G. Berry, "Programming with Concrete Data Structures and Sequential Algorithms," *Proc. ACM Conf. on Functional Programming Languages and Computer Architecture* (1981).

10. G. Berry, "Some Syntactic and Categorical Constructions of Lambda-calculus Models," Rapport INRIA 80 (D).

11. G. Berry and P.L. Curien, "The Kernel of the Applicative Language CDS: Theory and Practice," in *This Volume* (1982).

12. G. Berry and P.L. Curien, "Sequential Algorithms on Concrete Data Structures," *Theoretical Computer Science* 20, pp.265-321 (1982).

13. G. Berry and J.J. Lévy, "Minimal and Optimal Computations of Recursive Programs," *Journal of ACM* 26(1), pp.148-175 (Jan. 1979).

14. G. Berry and J.J. Lévy, "A Survey of Some Syntactic Results in the Lambda-calculus," in *Proc. Ann. Conf. on Mathematical Foundations of Computer Science, Olomouc, Tchecoslovaquia*, Lecture Notes in Computer Science 74, Springer-Verlag (1979).

15. G. Berry, *Stable Functions on Complete and Bicomplete Partial Orders*, To appear.

16. B. Courcelle and J.C. Raoult, "Completion of Ordered Magmas," Fondamenta Informatica, Poland (D).

17. P.L. Curien, "Algorithmes Séquentiels sur structures de données concrètes," Thèse de Troisième cycle, Université Paris VII (D).

18. P.L. Curien, "Algorithmes Séquentiels et Extensionnalité," L.I.T.P. report, Univ. Paris VII (D).

19. P.L. Curien, "Full Abstraction for a Calculus of Sequential Algorithms," LITP report 82-5, University Paris VII (D).

20. M.C.B. Hennessy and G.D. Plotkin, "Full Abstraction for a Simple Parallel Programming Language," in *Proc. MFCS 79*, LNCS 74, Springer-Verlag (1979).

21. R. Hindley and G. Longo, "Lambda-calculus Models and Extensionality," *Zeitschrift fur Mathematische Logik* 26, pp.289-310 (1980).

22. G. Huet, "Résolution d'Equations dans les Langages d'Ordre 1,2...ω," Thèse de Doctorat d'Etat, Université Paris VII (D).

23. G. Huet and J.J. Lévy, "Computations in Non-Ambiguous Linear Term Rewriting Systems," Rapport IRIA-LABORIA 359 (D).

24. G. Huet and D.C. Oppen, "Equations and Rewrite Rules: A survey," in *Formal Language Theory*, ed. R. Book, Academic Press (1980).

25. G. Kahn and G. Plotkin, "Structures de données concrètes," Rapport IRIA-LABORIA 336 (D).

26. C.P.J. Koymans, "Models of the Lambda Calculus," Preprint NR 223, University of Utrecht (D).

27. J.J. Lévy, "An Algebraic Interpretation of the $\lambda-\beta-\kappa$-calculus and an Application of a Labeled Lambda-calculus," *Theoretical Computer Science* 2(1), pp.97-114 (1976).

28. J.J. Lévy, "Réductions Correctes et Optimales dans le Lambda-calcul," Thèse de Doctorat d'Etat, Université Paris VII (D).

29. J. Lambeck, "From Lambda-Calculus to Cartesian Closed Categories," pp. 375-402 in *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, ed. J.P Seldin and J.R. Hindley, Academic Press (1980).

30. A.R. Meyer, "What is a Lambda-Calculus Model?," MIT Lab. of Computer Science MIT/LCS/TM-171 (D).

31. R. Milner, "Models of LCF," Computer Science Memo AIM-186/CS 232, Stanford University (D).

32. R. Milner, "Fully Abstract Models of Typed Lambda-Calculi," *Theoretical Computer Science* 4(1), pp.1-23 (1977).

33. M. Nielsen, G. Plotkin, and G. Winskel, "Petri Nets, Event Structures and Domains," *Theoretical Computer Science* 13(1), p.85,108 (1981).

34. G. Plotkin, "LCF Considered as a Programming Language," *Theoretical Computer Science* 5(3), pp.223-256 (dec. 1977).

35. G.D. Plotkin, "A Structural Approach to Operational Semantics," DAIMI FN-19, University of Aarhus (D).

36. V.Yu. Sazonov, "Sequentially and Parallelly Computable Functionals," in *Proc. Symp. on Lambda-Calculus and Computer Science Theory*, Lecture Notes in Computer Science 37, Springer Verlag (1975).

37. D.S. Scott, "Relating Theories of the Lambda-Calculus," pp. 403-450 in *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, ed. J.P Seldin and J.R. Hindley, Academic Press (1980).

38. M.B. Smyth, "The Largest Cartesian Closed Category of Domains," Dept. of Comp. Sci. report CSR-108-82, Univ. of Edinburgh (D).

39. M.B. Smyth and G.D. Plotkin, "The Category-theoretic Solution of Recursive Domain Equations," *Proc. 18th. Symp. on Foundations of Computer Science, Providence, R.I.* (1977).

40. J.E. Stoy, *Denotational semantics*, MIT Press, Cambridge, Mass. (1977).

41. J. Vuillemin, "Proof Techniques for Recursive Programs," Ph.D. Thesis, Stanford University (D).

42. C.P. Wadsworth, "The Relations Between Computational and Denotational Properties for Scott's D^{∞} Model of the Lambda Calculus," *SIAM Journal on Computing* 5(3), pp.488-522 (1976).

43. C.P. Wadsworth, "Approximate Reductions and Lambda-Calculus Models," *SIAM Journal on Computing* 7(3), pp.337-357 (1978).

44. M. Wand, "Fixed-point Constructions in Order-Enriched Categories," Research Report TR 23, Indiana University (D).

45. P.H. Welch, "Continuous Semantics and Inside Out Reductions," pp. 122 147 in *Proc. Symposium on Lambda calculus and Computer Science Theory, LNCS 37*, ed. C. Boehm, Springer Verlag (1975).

46. G. Winskel, "Events in Computations," Ph.D. Thesis, Univ of Edinburgh (D).

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique